Managing Free Software Projects
Abstract

Developing as free software is a powerful and popular way to create software. Many successful projects such as Apache, GNU/Linux and Mozilla Firefox were created as free software. In this degree project we will take a close look at how to manage free software projects. A free software project named Shaskel will also be set up as a practical example.

Keywords

free software, licenses, project management, project management tools
Acknowledgements

- Jesper Andersson - For supervising the thesis.
- Hein-Pieter van Braam - For teaching me a lot about free software.
- Shaskel project members - For helping out with the Shaskel project.
  - AllegedlyDead
  - Club
  - Elsamuko
  - Fisk
  - Myotonic
  - Rainbot
  - TMM
## Contents

1 Introduction  .................................................. 1
   1.1 Goals .................................................. 1
   1.2 Structure ............................................. 1
   1.3 Free software ......................................... 1

2 Project website .............................................. 3
   2.1 Free software hosting platforms ....................... 3
       2.1.1 SourceForge ........................................ 3
       2.1.2 Github ............................................... 4
       2.1.3 Google Code ........................................ 4
       2.1.4 CodePlex ............................................ 4
       2.1.5 Launchpad ........................................... 4
   2.2 Project management applications .................... 5
       2.2.1 Redmine ............................................. 5
       2.2.2 Trac ................................................ 6

3 IRC Channel .................................................. 8
   3.1 Discussions on IRC ...................................... 8
   3.2 IRC networks ............................................ 9
   3.3 IRC Clients ............................................. 9
   3.4 Setting up a channel .................................... 9
       3.4.1 Using multiple channels .......................... 10
   3.5 IRC Bots ................................................. 10

4 Version control systems ..................................... 12
   4.1 Centralized Version Control ........................... 12
   4.2 Distributed Version Control ........................... 13

5 Licenses ......................................................... 14
   5.1 Free software definitions .............................. 14
       5.1.1 Free Software Foundation .......................... 14
       5.1.2 Open Source Initiative ............................. 15
   5.2 GPL-like licenses ....................................... 15
       5.2.1 GPL .................................................. 15
       5.2.2 LGPL ................................................ 16
       5.2.3 AGPL ............................................... 16
   5.3 BSD-like licenses ....................................... 16
       5.3.1 BSD .................................................. 17
       5.3.2 Other ................................................ 17
   5.4 Creative commons ....................................... 17
   5.5 Public domain and other licenses ..................... 18
       5.5.1 Beerware ............................................ 18
       5.5.2 Unlicense ........................................... 19
   5.6 Picking a license ....................................... 19
       5.6.1 Software ............................................. 19
       5.6.2 Artwork ............................................. 19
6 Making progress

6.1 Employees .................................................. 20
6.2 Google summer of code ................................. 21
6.3 Donations ..................................................... 22
   6.3.1 Subscribe ................................................. 22
   6.3.2 Buying .................................................. 23
   6.3.3 Donate .................................................. 23
   6.3.4 Issues .................................................. 23
6.4 University projects ...................................... 23
6.5 Fun, mastery and kindness ............................. 24

7 Handling Issues ............................................. 25

7.1 Issue types ................................................ 25
   7.1.1 Bug reports ............................................. 25
   7.1.2 Feature requests ...................................... 25
   7.1.3 Patches ............................................... 26
7.2 Getting issues resolved .................................. 26
   7.2.1 Ubuntu Hug Day ...................................... 26
   7.2.2 Ubuntu 5-A-Day ....................................... 26
   7.2.3 Bounties ............................................... 27

8 Forks ........................................................... 29

8.1 Reasons for forking ...................................... 29
8.2 Forked projects .......................................... 30
   8.2.1 XFree86 and X.Org .................................... 30
   8.2.2 Compiz and Beryl .................................... 30
   8.2.3 OpenOffice.org and LibreOffice .................... 30

9 Profiting from free software ............................ 32

9.1 Licenses ...................................................... 32
9.2 Relicensing ................................................ 32
9.3 Services ..................................................... 33
   9.3.1 Support ................................................. 33
   9.3.2 Storage ................................................ 33
   9.3.3 Education ............................................. 34
9.4 Hardware .................................................... 34

10 Shaskel ....................................................... 36

10.1 Setting up the project .................................. 36
   10.1.1 Website ............................................... 36
   10.1.2 IRC Channel ........................................ 36
   10.1.3 Version control system ............................ 36
   10.1.4 Backup ............................................... 37
10.2 Storyline .................................................. 37
10.3 Finding contributors .................................... 37
   10.3.1 Reddit posts ........................................ 38
10.4 License ..................................................... 38
10.5 Tools ......................................................... 38

11 Conclusion .................................................. 39
1 Introduction

Developing as free software is a powerful and popular way to create software. Many successful projects such as Apache, GNU/Linux and Mozilla Firefox were created as free software. In this degree project we will take a close look at how to manage free software projects. A free software project named Shaskel will also be set up as a practical example.

1.1 Goals

The goal of this degree project is to provide a guide for how to set up and manage free software projects. The goal is also to look for various ways to profit from free software. The question this report aims answer is how free software projects are managed.

1.2 Structure

In order to provide an answer to how free software project are managed we will look at how previously successful free software projects have been managed. We will also look at some helpful project management tools and various free software licenses. The report has been, as requested by the supervisor Jesper Anderson, written in a manner similar to a tutorial. Each section is rather independent but will together provide essential knowledge in how to manage free software projects.

First we will take a look at some useful tools for managing a free software projects, namely project websites (section 2), IRC channels (section 3) and version control systems (section 4). After that we will look at some popular free software licenses (section 5). Then we will look at how to manage projects, specifically how to make progress (section 6), how to handle an issue tracker (section 7) and how projects are forked (section 8). We will then proceed with discussing how to profit from free software projects (section 9). Finally, we will set up a complete free software project called Shaskel (section 10) followed by the conclusion.

1.3 Free software

The free software foundation defines free software with the following “four essential freedoms”, quote[9]:

- “The freedom to run the program, for any purpose (freedom 0).”
- “The freedom to study how the program works, and change it to make it do what you wish (freedom 1). Access to the source code is a precondition for this.”
- “The freedom to redistribute copies so you can help your neighbor (freedom 2).”
- “The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.”
Some people may agree while others disagree with “free software” being defined by these “four essential freedoms”. I think they capture the essence of “free software” well and we will use them as the “free software” definition for this report.

The name brings up another issue, some may call it “free software” while others call it “open source software”. Some may even combine these terms into “free open source software” or even “free libre open source software”. All these different names have a slightly different meaning, for this report we will simply use the term “free software”.

Do note that “free” means “free as in freedom” (libre) and not “free as in beer” (gratis). That is, free software can sometimes cost money but the freedom of the user will always be preserved. This is discussed in more detail in the section 9.
2 Project website

Perhaps the most important tool to a free software project is a project website. The website is the first place where developers and users come looking for information, it also acts as a gateway to other tools such as the version control system, issue tracker, wiki, forums, IRC channel and so on.

The websites can come in many shapes and forms. Some projects have a full-fledged project management system while others just have a single HTML page with some links. There are also various ways to create such a website, the sites can be hosted on a private web server or on a free software hosting platforms such as SourceForge or GitHub.

This section describes the various ways project website can be created. It starts with describing public free software hosting platforms and later systems that can be deployed on private servers.

2.1 Free software hosting platforms

There are many websites on the Internet that will host free software projects for free. By letting others take care of the project hosting it is possible to save time and resources. It could also be the case that a person starting a project does not have access to any servers for where to host the project. There are quite a few of these websites, some will be described in this section, these include:

- SourceForge
- GitHub
- Google Code
- CodePlex
- Launchpad

Other project hosting websites will not be discussed in detail are Bitbucket, GNA, Savannah and BerliOS. There are most likely many more project hosting websites that this report has neglected to mention.

Selecting one of these platforms is a matter of taste, they are all however different in various aspects. These differences will be described in more detail throughout this section.

The websites mentioned above do have one thing in common, they are all complete platforms for hosting free software projects. In other words, not only do they supply projects with a page for information, they also supply projects with version control, issue trackers, wikis and so on.

2.1.1 SourceForge

SourceForge was launched in 1997 and hosts as of March 2011 over 260 000 free software projects [18]. The platform features version control, issue tracker, MySQL database, web pages, file hosting and a subdomain for each project [7]. SourceForge support many different version control systems including CVS, SVN, Git, Bazaar and Mercurial. Starting a project on SourceForge is free, the project must however be a public free software project.
2.1.2 Github

GitHub is a relatively new hosting platform launched in 2008 [67]. As the name implies, the platform is built around Git and does not allow for using any other version control system. Along with the version control system it features issue trackers, wikis and web pages that can be edited via a Git repository. GitHub is free to use for public open source projects but can for a price also be used to host private proprietary projects. Although the GitHub server software is not free software it is still possible to license and run it on a local server.

Along with anonymous cloning URLs there is also a feature for exporting and downloading repository snapshots without using Git. GitHub attempts to extend the distributed version control system by adding social networking features. These features include feeds, followers and graphs that show how repositories are cloned and worked on.

The user interface includes tools for cloning other repositories, this means that it is very easy to start contributing to other projects. After cloning a repository and pushing commits into the clone pull requests can be created in the issue tracker of the original repository. The owner of the original repository can then accept these requests and pull the commits into the original repository.

2.1.3 Google Code

Google code was launched by Google in 2006 with searchable code as its main feature[2]. The platform features a source code tracker, search engine, issue tracker, version control and a version control browsers. Google Code’s search features makes it possible to search in the wikis, issue trackers and source code repositories of all the projects hosted at Google code.

Google code has support for two version control systems; Subversion and Mercurial [28]. The repositories can be browsed with the built in source code browser. If a project uses another version control system the source code browser page can be edited to just contain a link to the repository. However, using external repositories prevents the source code from becoming searchable.

2.1.4 CodePlex

CodePlex is a project hosting platform created by Microsoft in 2006[3]. The platform features version control, discussions, wiki, issue tracker, downloads and news feeds.

Some parts of the platform has been released as free and open source software. As of March 2011 CodePlex client, WikiPlex, SvnBridge and Diffplex has been released as free software and more parts are to be released in the future.

2.1.5 Launchpad

Launchpad was released by Canonical (the company behind Ubuntu) in 2004 and was made available as free software in 2009[55]. The platform contains a variety of tools for aiding development including a forum, issue tracker, version control and translations[36]. Unlike the other platforms it does not feature a wiki and is pretty limited when it comes to customizing the project pages. Launchpad only allows for using Bazaar (also created by Canonical) as the version control system.

What makes Launchpad unique is its support for PPAs (Personal Package Archive). The PPA system allows developers to upload archives containing source code and
compilation rules to Launchpad. The Launchpad servers will then use those rules to create installable packages of the software for Debian-based Linux distributions. Users of the software can then add the PPA to their package manager which makes the software easy to install and update.

2.2 Project management applications

Instead of using one of the project hosting platforms hosting the project yourself is also an option. This can sometimes be preferable as you gain more control over the project. There is however also a risk to this, make sure to backup all the project data thoroughly and often if you decide to host the project yourself.

Some of the software running on the software hosting platforms described in the previous section are free software. This means that they can with some work be downloaded, installed and run on private servers. Some may not be easy to install but should still function well once installed.

There are also project management applications which have been designed to be downloaded, installed and run on private servers. The applications we will take a closer look at includes:

- Trac
- Redmine

These applications have some things in common; they are all free open source project management software that have web-based user interfaces.

2.2.1 Redmine

Redmine is a free and open source project management tool written in Ruby using the Ruby on Rails framework[44]. The tool works on multiple platforms, web servers and databases.

What makes Redmine interesting is that it is extremely feature rich. Below are the features as of version 1.0[44].

- **Support for multiple projects** - Can run as a single instance on the server but manage multiple projects in the same instance.
- **Issue tracking** - Complete issue tracker with types, subjects, descriptions, comments, status, priorities, assignments, dates, estimated time required, progress percentage, attachments, watchers, previews and so on.
- **Charts and calendar** - The gantt chart and calendar gives developers a quick overview of the project and helps with the planning and scheduling of the different tasks.
- **File storage** - Two different sections made only for storing files. One of them is Documents and the other one is files. Documents is, as the name implies, made for storing project related documentation. Files is more general and can for example be used for distributing stable releases of the software that is being developed.
- **News** - A small blog-like feature for releasing various project-related announcements.
- Feeds - Generation of RSS feeds for many different parts of the project. This makes it easy to stay updated without getting mail accounts flooded.

- E-mail notifications - Users can chose to receive e-mail notifications for various project events.

- Wiki - Each project gets its own wiki for documentation.

- Forum - Each project had a forum, complete with sections, file attachments and stickies.

- Time tracking - Allows developers to log the time he or she spent on the project by adding different activities to the time tracking system. Tasks added to this system has a date, time, type, description and can be linked to a specific issue. Redmine can create time tracking reports as well as export the data to the CSV format.

- Version control - Support for viewing a wide array of version control repositories. Supported version control systems includes Bazaar, CVS, Darcs, Git, Mercurial and SVN.

- User management - A very flexible system for managing users and their permissions. User registration and verification can be turned on or off by administrators. It is also possible to set projects (or a whole Redmine instance) as private which means that only registered members or members with a specific role will be able to access projects. A user can for each project get one or more customizable roles. With each role the user gains additional privileges such as uploading documents to the file server.

- Plug-ins - If the features described above are not enough it is possible to add more using plug-ins. Plug-ins can also be used to modify already existing features. One example of this is a plug-ins which forces Redmine to always hide e-mail addresses by default.

Redmine is the project management application used in the Shaskel project (see section 10.1.1).

2.2.2 Trac

Trac is a very popular (in 2011) free and open source project management tool. It was written in Python and is used by many projects including PulseAudio, jQuery, Pidgin, OpenMoko and Django[46]. The application was first released in October 2006 under a modified version of the BSD license.

Trac hosts one project per instance which means that a new instance of Trac will have to be configured for each project, this is however very likely to change in the future.

While Trac might not be as feature rich out of the box as Redmine (section 2.2.1) but can be extended via plug-ins. Below are a list of features as of version 0.12[16].

- Issue Tracker - Color-coded issue browser where each issue gets a color depending on its priority. Each issue has a priority, severity, owner, milestone, component, commend and multiple keywords. It is also possible to attach various files to issues or comments.
- **Version control** - Trac features a built-in version control viewer for SVN only, support for other version control systems can be added via Tracs’ plugin system. There are currently plugins for Git, Perforce, Mercurial, Darcs, Bazaar and Monotone[45].

- **Wiki** - Features file attachments, page histories and also an index which list every wiki page in that particular Trac instance. The wiki isn’t empty when Trac is first initialized but contains useful information about Trac and instructions on how to use it.

- **Milestones** - Milestones (viewable via “Roadmap”) can be used to set up goals for a project. A milestone is basically a collection of issues accompanied by a description for that particular milestone. The milestones usually have a due date for when all issues related to that milestone are supposed to be closed.

- **Timeline** - Shows a summary of what has changed in the project. The timeline shows changes to the source code repository, milestones, issues and wiki pages.

- **Plug-ins** - There are many different plug-ins available for Trac. Features that can be added includes a web administration interface, spam protection, build systems, code documentation (such as javadoc), file management, RPC, testing, forums, user management and various version control systems.
3 IRC Channel

IRC (Internet Relay Chat) is a protocol for directly chatting with other people in various IRC channels (chat rooms) or via private messages. IRC plays an important role in most larger free software projects. It allows people to instantly communicate with each other without waiting for others to reply to forum posts, e-mails or issue comments.

By using IRC users can discuss various topics with the project members. IRC can also be used to get direct help or support from the project members or even other users of an application. By chatting one also gets a chance to get to know the project members and get more involved with the project.

3.1 Discussions on IRC

IRC discussions are not very persistent. Due to its nature as a chat, discussions that occur on IRC are often only read by people who are present at the moment. This is true especially if the IRC channel is popular, active and has a lot of connected people.

By using channel loggers people who are not there will still be able to read the discussions. While this keeps the history of the channel well-preserved, project members are not likely to read through the logs unless they are looking for something specific. One reason for this is that the amount of text lines in an IRC channel for just a single day can be huge. For example, after being in the Freenode for just 5 minutes I had already seen 16 lines of text (excluding people entering or leaving the channel).

When a channel is too active a discussion might get “drowned” in all the other discussions (solutions for this will be presented in following sections). However, if a channel is too inactive there might not be anyone present to discuss with. One important thing to note is that the amount of users may not have anything to do with whether a channel is active or not. A channel might have a lot of people in it, but very few of these might actually be awake. It is also very common that users leave their IRC channel online even when they are not present. For example, when I asked a question about where the money donated to Ardour goes to I got the answer:

<flower> you better ask this a few hours later, when usa is awake

An couple of hours later I got the answer I wanted from Ardour’s team members and main developer. Their clients had been connected all along even though they were actually away. Due to different time zones, work hours and other reasons different people will be active at different times. It is therefore important to be patient when chatting on IRC. If questions are not answered directly, then one should simply wait for an answer and ask the question again after that a few hours have passed. While waiting might seem to defeat the purpose of being able to contact people directly, one will still get a more interactive discussion when someone actually responds.
3.2 IRC networks

Just like there multiple channels and users on IRC there are also multiple “networks”. A network is simply a server (or a collection of servers) to which users can connect and chat on. Each of these networks have their own set of channels, features, users and rules. This can be compared to forums online: there are multiple forums on the Internet, each of them with their own topics, features, users and rules.

Just like with forums anyone can set up and configure their own IRC networks, this will not be covered in this report. We will however briefly look at two networks used by multiple free software projects:

- **Freenode** - `irc.freenode.net` - Primarily aimed at providing “discussion facilities” for free and open source projects[19]. Its user base peaked at about 70000 users. Freenode also has support for SSL encryption. At the time of writing Freenode hosts around 10500 different channels with the largest channel being #Ubuntu with 1600 users.

- **GIMPnet** - `irc.gimp.org` - GIMPnet is a network primarily used by projects related to GNOME (the Desktop Environment).

3.3 IRC Clients

An IRC client is needed to take part in discussions on IRC. There are many different IRC clients, some are stand-alone while others built into instant messenger clients, some have a GUI while others use the command line, some are free software while others are proprietary. Below are a few free software clients:

- **IRSSI** - `http://irssi.org/` - Cross-platform command line client licensed under the GPLv2 license.

- **Pidgin** - `http://pidgin.im/` - Cross-platform multi-protocol instant messenger client with IRC support. Has a graphical user interface and is released under the GPLv2. There is also a command line version called Finch.

- **XChat** - `http://xchat.org/` - UNIX only client with a graphical user interface. XChat is released under the GPLv2 license. A proprietary port of XChat is available for Windows.

There are also a few web-based clients which turns almost any browser into an IRC client without having the need for plug-ins or extensions. One of these web clients is called qwebirc (`http://qwebirc.org/`) which is released under the GPLv2 license. There are a few publicly available instances of qwebirc, one of is hosted by the Freenode network at `http://webchat.freenode.net`.

3.4 Setting up a channel

Creating an IRC channel is easy on most IRC networks. In many IRC networks (Freenode included) it is enough to join a channel which does not yet exist nor is already registered to someone else. When such a channel is joined it will automatically be created and the user will gain operator (administrator) privileges to it.

If the user then leaves the channel it will be removed. To prevent that from happening the user must first have a registered user account on the network. Then,
the channel must be registered using that user account. Registering user accounts and channels can often be done by chatting to special IRC bots provided by the network, these bots are called NickServ and ChanServ on Freenode. An alternative way is to always have users or bots present in the channel, this is sometimes the only way as some networks may not provide services for registering channels.

After registering the channel it will no longer disappear when it becomes empty. It will also be possible for the founder of the channel to ask the network to give him or her operator privileges. The founder can even give other users the ability to gain operator privileges by talking to the network services.

Having multiple people with operator privileges in a channel can be important. This is because it is not uncommon for IRC channels to be flooded by spam bots or malicious users. The IRC network staff on Freenode is helpful when it comes to preventing flooding or helping out with other issues. Any user can join the #freenode channel and ask for help there.

3.4.1 Using multiple channels

It can sometimes be a good idea to use multiple IRC channels for the same project. Multiple channels allows for the different kinds of discussions in a project to be divided up into the smaller channels. This can make the discussions more focused which in turn makes them easier to follow and participate in. An example to this is the Warzone 2100 free software game. Warzone 2100 has one channel dedicated to development-related discussions, one channel for finding people to play with and a third channel for other discussions and support.

3.5 IRC Bots

IRC bots are applications that are present in many IRC channels and provide some kind of service. These bots can either be hosted by the network itself, communities or even individual users. The bots can be helpful to free software projects in many ways, this report briefly covers a few of these cases below.

- **Channel loggers** - Perhaps the most common kind of bot. The channel logger simply monitors and stores the activity in a channel to log files. Users with access to these logs can then read or search through them in case they need to access any previous discussions that occurred in the channel. The logs can also be used to generate statistics. The statistics can then be used to, for example, find out at what time of the day users are most active or who the most active user in a certain channel is.

- **Title fetchers** - Title fetchers only job is to monitor the channel for links to websites. When such a link is found the bot downloads the page and extracts the title which it announces to the channel. This helps by showing the users in a channel what the link contains and what website it leads to without them having to actually open the link.

- **FAQ** - In larger channels such as #Ubuntu different people are bound to ask the same questions over and over again. An FAQ bot (such as the one in #Ubuntu) has answers to many of these common questions. The answers will be announced by the bot when told to do so via a text message. This way
common questions can be answered quickly without people having to manually write or copy and paste the answer.

- **Change announcers** - Announces changes to a project in the project’s IRC channel. For example, if someone committed changes to the project’s source code repository, then the bot would announce these new commits in the IRC channel. Some of these bots can also announce changes in parts of the project management platform such as in issue trackers. Announcing changes in this way can help project members and users keep track on what’s going on within the project.

Note that one bot may perform several of these tasks at the same time. It could even be that a regular client used by a user for chatting performs one or more of these tasks as well. An example to this is that most IRC clients provide a way for users to log the activity in channels where they are present.
4 Version control systems

Another very important part of managing a free software project is to have a publicly available and up-to-date version control repository. Without one the project members would have to manually send updated versions of the project files to each other. Users would have to rely solely on releases and it would be difficult to integrate patches from people who are not directly involved in the project. Thankfully, the version control systems solve all of these issues for us.

Version control systems keep tracks of files and changes within projects. Project members can use these systems for storing and sharing project-related files such as source code. When a project member changes a file locally these changes can then be uploaded to the version control “repository”. Other members will then be able download these changes and all other changes “committed” by other project members. As the version control system keeps tracks of all changes made to the files in the repository, one can look at the history to see what has changed and even revert local files back to any previous version.

There are many benefits with actively using version control within a project, including:

- **Convenience** - The version control system takes care of transferring changes between project members and users. Everyone will always be able to easily update to the latest version of the project files.

- **History** - Everyone will be able to view and use any previous version of the application. It is easy to see what has changed, who made specific changes and how the software develops over time. It may even be useful for locating regressions as one can easily get an overview of all changes to each and every part of the software.

- **Contributions** - People contributing to a project can always do so for the latest version of the code. Instead of creating patches for a (possibly old) release which may then have to be modified, either because of conflicts or because the software’s behaviour has changed, they can be applied directly to the project.

- **Testing** - The latest version of the project can always be tested. That way bugs have a chance of being discovered before a releases is made.

There are two major groups of version control systems, centralized and distributed. We will take a look at both groups and a couple of version control systems within each of these groups.

4.1 Centralized Version Control

Centralized version control systems have, as the name implies, a centralized repository. Project members and users can download the files contained in the repository as well as update them when new changes have been uploaded. Project members work towards the repository server which is queried for every version control-related action taken by the user. Examples to these actions are downloading changes, uploading changes, looking at the project history and reverting to previous versions.

There are quite a few different centralized version control systems, some free software ones include:
• **CVS** - CVS stands for Concurrent Version Control system and was created as an RCS (Revision Control System) clone with some “significant advantages”[1]. The main advantage over RCS was that the system had a client/server architecture which turned it into a centralized version control system. While CVS was popular some time ago it is rarely used in new free software projects and has been superseded by Subversion.

• **Subversion (SVN)** - Subversion (or SVN) was meant to be “CVS done right” and was created to take over the CVS user base[20]. Subversion is a bit simpler to use and does not, for example, require the user to manually log in as anonymous to download files contained in a repository.

### 4.2 Distributed Version Control

Instead of always working towards repository servers distributed version control system will clone entire repositories, store them locally and work towards the local copies. When a distributed repository is cloned all files and the complete history of those files will be stored locally. This means that one does not need to contact the repository server to, for example, revert to an older version of the project or browse the repository history. To download changes that others have uploaded one will of course still need to connect to a repository server and fetch them from there.

Unlike centralized version control systems changes do not immediately have to be uploaded to the remote repository server in order to create a new “revision” (version). Instead, one can store many new versions in the local repository before “pushing” (uploading) one or more of these changes to the server. As each person working on the project has a complete copy of the repository everyone may not necessarily upload to and download from the same remote repository. One can for example upload or download changes directly to another project members’ local or remote repository.

Forking becomes much easier with distributed version control systems as the history of the repository is retained in all clones. This is possible to do with centralized version control systems as well, but one would need to download each revision from the server and reconstruct the repository that way.

• **Git** - Git was created by Linus Torvalds because the Linux kernel was in need of a new distributed version control system[29]. Contrary to SVN git was not supposed to be “CVS done right”, it was instead created with the “what would CVS not do” mindset.

• **Mercurial (hg)** - Mercurial is a version control system designed to be easy to learn and use while still being fast and powerful[12]. Mercurial is used in a way very similar to that of Git.
## 5 Licenses

Selecting the right license for your project is very important. The licenses will decide what users and developers will be able to with the project. The licenses will also decide where and how the project can be distributed, used and whether or not it can be sold.

There are many different licenses to choose from. Software licenses are generally divided into two groups; GPL-like and BSD-like licenses. Some licenses (such as creative commons) are specifically made for assets (artwork and other data). There is also the public domain and some public domain-like licenses such as Beerware and Unlicense.

As with Shaskel (see section 10.4), multiple licenses can be used for different parts of the same project. This is common for applications with a lot of artwork such as games. One could also argue that using a library with a license different from the one used in the project means that the project is using multiple licenses.

It is also possible to use multiple licenses for the same parts of a project. Doing so is called dual-licensing or multi-licensing and is done mostly to make the project more compatible with other software. An example to this is OSS (Open Sound System) which is licensed under the GPL, BSD, CDDL or 4Front commercial license. OSS is licensed this way to make it more compatible with the kernels of multiple operating systems. For example, because it is dual-licensed it is possible to distribute it in the Linux kernel source tree as GPL or in the OpenSolaris kernel source tree as CDDL.

We will take a look at the following licenses:

- **GPL-like** (GPL, LGPL, AGPL)
- **BSD-like** (BSD, MIT, EPL)
- **Creative commons** (BY, SA, NC)
- **Public domain-like** (Public domain, Beerware, Unlicense)

But first we will look at how different organizations define free software.

### 5.1 Free software definitions

There are two major organisations that are attempting to define what “free” or “open source” software is, namely the Free Software Foundation and the Open Source Initiative. Both these definitions provide a good way of seeing whether a license is “good enough” for use within a free software project. If both organisations approve of a license then one can be quite confident in that the license is “good enough” when it comes to freedom for both users and developers.

#### 5.1.1 Free Software Foundation

As discussed in the introduction (section 1.3) the Free Software Foundation defines free software with four essential freedoms, quote[9]:

- “The freedom to run the program, for any purpose (freedom 0).”
• “The freedom to study how the program works, and change it to make it do what you wish (freedom 1). Access to the source code is a precondition for this.”

• “The freedom to redistribute copies so you can help your neighbor (freedom 2).”

• “The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.”

A more detailed description and motivations for these essential freedoms can be found at http://www.gnu.org/philosophy/free-sw.html.

A list of licenses, both those considered free and those considered non-free, by the Free Software Foundation can be found together with motivations at http://www.gnu.org/licenses/license-list.html.

5.1.2 Open Source Initiative

The definition for open source software by the Open Source Initiative is quite lengthy and thus will not be included in this report. Although more detailed, the definition is somewhat similar to the Free Software Foundation’s “four essential freedom” in the section above. For the complete open source definition by the Open Source Initiative, see http://www.opensource.org/docs/osd.

A list of licenses considered open source by the Open Source Initiative and be found at http://www.opensource.org/licenses/.

5.2 GPL-like licenses

GPL stands for General Public License. The original GPL license was released by the free software foundation in 1989. Notable GPL-licensed software includes Linux, Redmine (section 2.2.1), GNOME, Ardour and BASH.

GPL-like licenses are designed to keep application free software. This means that if an application is licensed under the GPL then the source code for that application must always be available.

The source code must not necessarily be distributed together with binaries (files generated from the source code) or devices containing software licensed under the GPL. However, there must always be a note describing that the software contains parts licensed under the GPL together with a description on how to acquire the source code.

5.2.1 GPL

The “regular” GPL license was created to solve to force distributors of free software to also make the source code available. The license also tries to solve the problem where distributors would combine the free software with parts that had restrictions on usage and distribution. This was done by only allowing software under the GPL to be distributed together with software licensed under a more permissive license.

There are three different versions of the GPL where each new version adds more restrictions meant to guarantee the freedom of the users. The new restrictions are
added because the way that we use hardware and software change over time as we create both new hardware and software.

An example to one of these new restrictions added in the GPL version 3 is protection against tivoization\[5\]. Tivoization means that the software running in a device can not be replaced due to some restrictions (for example digital signatures). Because of the tivoization, even if the source code for an application was available modifying it would be meaningless as the device would refuse to execute the modified version. Software licensed under the GPLv3 can thus only be used in devices that would accept a modified version of said software.

5.2.2 LGPL

LGPL (Lesser GPL) makes free software more compatible with proprietary (closed source) software. The LGPL was created for libraries in 1991 and was then called Library GPL before changing name to Lesser GPL for philosophy reasons. The license is basically the GPL license but without the part which only allows software released under the GPL to be distributed together with software under a more permissive license.

The effect of this is that software licensed under the LGPL can be used by proprietary software as well. For example, the SDL (Simple Directmedia Layer) and be used by and distributed with a proprietary application as long the LGPL notice is included as well. If this would have been GPL then the proprietary application would have had to be open source in order to use SDL.

As LGPL allows for usage together with proprietary software there are some perhaps unexpected side effects when it is used for other software than libraries. New features can be added to the LGPL licensed software without the source code of these new features having to be available. This is done simply by linking a proprietary part with the LGPL licensed software just like you would with a library.

5.2.3 AGPL

AGPL, also called Affero GPL, was designed specifically for web applications such as Redmine (section 2.2.1). While the GPL makes sure that the source code for free software is available whenever the software is distributed one does not have to distribute the source code when only using an application. This creates a problem for web applications as the applications can run on a web server without actually distributing the application itself. AGPL solves this problem by also forcing users of an application to make the source code available.

The users will most likely not have to put in much effort to comply with AGPL licensed applications as many of them includes a notice and links to the source code by default.

5.3 BSD-like licenses

Contrary to GPL-like licenses the BSD-like licenses allows for distributing the applications without the source code. The BSD-like licenses aims to keep options open and do not has as many restrictions on distribution as GPL.

Because software under BSD-like licences can be distributed without source code it may be more interesting for companies. Companies can thus make these applications closed source as long as they attribute the author and include a copy of
the license. It is also possible (much like LGPL in section 5.2.2) to extend the functionalities of software without releasing the source code for these modifications.

5.3.1 BSD

The BSD license states the following, quote:

- “Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.”

- “Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.”

- “Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.”

Together with those three clauses there is a short text about who the copyright belongs to. There is also a section disclaiming any warranty or responsibility for damages caused by using the BSD-licensed software. As you can see above, the only thing the BSD license demands is attribution to the author and that the authors can not be used to promote derived software.

Some notable software using the BSD license includes the BSD kernel, Open Sound System and Chromium.

5.3.2 Other

There are a few licenses that are similar to the BSD-license in the way that they allow for binaries to be distributed without supplying the source code. These kinds of licenses are often called BSD-style licenses.

- MIT - Massachusetts Institute of Technology
- Apache 2.0 - Apache License
- PSFL - Python Software Foundation License

Some notable software using these licenses are Ruby on Rails, Mono, Apache HTTP server and Python.

5.4 Creative commons

Creative commons licenses were designed for authors and artists to license media such as text, images, audio, video and so on. The creative commons licenses allows the author to pick and choose the terms of the license and add various restrictions to it.

The following restrictions can be added to the licenses:

- Attribution (by) - The original author must be credited when the work is distributed. This is true even if the work has been modified by others.
- **Noncommercial (nc)** - The work must not be used for commercial purposes. This can cause a number of problems when distributing the work, see section 9.1.

- **No derivative works (nd)** - The work must not be modified.

- **Share-alike** - The work and modified version of it must be distributed under the same license as the original work.

The creative commons website has a page where one answer a few questions about what a license should allow. When that is done the website will provide the user with a customized license by combining the Attribution, Non-commercial, No derivative works and share-alike restrictions. The user gets a link to a website where the license text is available along with small images and banners showing the terms of the license.

There are many places where one can find creative commons licensed works, including Jamendo (music, http://www.jamendo.com/), Freesound (sound effects, http://www.freesound.org/) and Texture Warehouse (textures, http://www.texturewarehouse.com/gallery/).

### 5.5 Public domain and other licenses

When an author releases something into the public domain it *basically* means that the author disclaims copyright for the work. In other words, the users of the work are free to do anything they want to it and use it in whatever way they please. Stating that the author disclaims copyright of a work may be better than stating that is under public domain. This is because the laws of some countries may not acknowledge the term ‘public domain’.

There are a few licenses similar to public domain. These licenses are used by people who generally do not care about for what or how their work is being used but are glad that it is. Two such licenses are Beerware and Unlicense.

#### 5.5.1 Beerware

Beerware was created by Poul-Henning Kamp[4] who just wanted to give software away without complicated licenses, Poul Henning writes on his website (http://people.freebsd.org/~phk/):

> "I have had it with lawyers trying to interpret freedom. If I write software which I intend to give away, I don’t want to have to stick several pages of legalese on it to make sure nobody exploits it or any such metastable. If I have decided that I’ll give away some code I’ve written, I going to give it away, period, none of this ‘unless it is worth a million to somebody’ rubbish."

The license itself states that anyone is free to do whatever they want with the software as long as the author is attributed (similar to BSD and CC SA licenses). Below is a complete copy of the Beerware license:

> "THE BEER-WARE LICENSE’ (Revision 42):
phk@FreeBSD.ORG wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day,
and you think this stuff is worth it, you can buy me a beer in return
Poul-Henning Kamp”

The license sticks out from other licenses with the part about users buying the
author a beer (which is how the license got its name). Do note that this license
has not approved by neither the Free Software Foundation nor the Open Source
Initiative.

5.5.2 Unlicense

Unlicense (http://unlicense.org/) more formally puts works under the public
domain. In countries that do not acknowledge public domain the license still provides
the same freedom to the users, It does this by stating the following:

“This is free and unencumbered software released into the public do-
main. Anyone is free to copy, modify, publish, use, compile, sell, or dis-
tribute this software, either in source code form or as a compiled binary,
for any purpose, commercial or non-commercial, and by any means.”

Additional text about copyright law and warranty follows the text above. As
one can see, the license first states that the work is released under the public domain
and then also what this implies.

5.6 Picking a license

As written previously, selecting the correct license(s) for you project is important.
This section aims to help with selecting a licenses for your project but are strictly
my own recommendations. Whichever license you may chose, make sure to take a
close look at the licenses you consider before making a final decision.

5.6.1 Software

• GPL-like (section 5.2) - When source code should always be offered when
  binaries are distributed.
  – AGPL (section 5.2.3) - When software must not be used in proprietary
    software and is meant to run on a remote machine (web application).
  – GPL (section 5.2.1) - When software must not used in proprietary soft-
    ware and is meant to run locally.
  – LGPL (section 5.2.2) - When it is allowed to distributing the software
together with proprietary applications.

• BSD-like (section 5.3) - When binaries can be distributed without offering
  the source code.

• Unlicense (section 5.5.2) - When no restrictions should be placed on usage
  or distribution.

5.6.2 Artwork

Creative commons when the artwork is an important and large part of the project.
Otherwise the artwork can go under the same license as the software which will
simplify distribution.
6 Making progress

Progress will not happen automatically just because a project is free software. Unless other individuals finds the project interesting or useful the project founders will most likely have to do most of the work themselves, at least in the early stages of a project. Starting up a free software project will probably not mean that a bunch of developers will show up and do all the work for you.

One very important part of getting contributors is to have working code. When there is some actually working software it is much easier for people to get involved with a project. People can see that the project is there, is working and that it may be worth contributing to it. It is also much easier to start contributing by sending small changes to the project rather than to just start building someone else’s software from the ground up. The small contributions in projects may lead to larger ones in the future as the contributor will be in contact with the project team. The contributor will then get to know the team and perhaps receive some good feedback on the contribution.

There are a number of reasons besides “just wanting to” that can motivate developers to work on a free software project. We will take a look at the following reasons of to why developers may be involved in a project:

- **Employees** - People employed at a company which for various reasons aims to develop some software as free software.

- **Google summer of code** - Students paid by Google to work on free software projects.

- **University projects** - Universities developing software as free software.

- **Donations** - Projects paying project members to develop free software via donations.

6.1 Employees

There are a large number of people working on free software projects because they are employed by a company to do so. Different companies have different reasons for funding free software projects, either for their own gain, for others or a mixture of both.

Below are a few examples of free software projects with employees from various companies:

- **Intel Linux Graphics** - Linux graphics drivers for Intel’s integrated graphics processors. These are developed by Intel[34] to make graphics hardware from Intel usable for users of the GNU/Linux operating system. These drivers are also free software which makes it possible to integrate them into the Linux kernel source tree and ship them with Linux distributions. In other words, they can work out of the box without users having to do any installation manually. Other vendors of graphics processors have only proprietary graphics drivers[54] that can not be shipped directly together with the Linux kernel and distributions but instead have to be installed manually. Proprietary drivers can also stop working with new versions of the display managers on a GNU/Linux system but can not be updated by the community due to them being closed
source. This can make Intel’s hardware much more interesting for Linux users and may therefore sell slightly better than some of their competitors.

- **Compiz** - Compiz is a window and compositing manager for UNIX-based operating systems running the X11 display manager. Compiz provides fancy visual effects such as transparency, animations and the relatively well-known “desktop cube” effect. Novell released Compiz (along with XGL) in 2006[66] with the goal to create a richer visual experience on the Linux desktop which they hoped would bring more users to Linux. As Novell provides services, support and products running on the GNU/Linux operating system, bringing more users to the platform would enable Novell to earn more money. Compiz was huge success and has no-doubt impressed many users (especially in 2006 when visual effects on PC desktops were sparse) and is today still used in some distributions.

- **MeeGo** - MeeGo is a GNU/Linux distribution mostly developed by Nokia, Intel, The Linux Foundation and the Maemo community. It is designed to run in smart phones, media phones, tablets, televisions and vehicles. Nokia is developing MeeGo to run on their future devices thus using free software to promote and sell their products.[53]

### 6.2 Google summer of code

Google Summer of Code (GSoC) is an annual event created by Google which offers students stipends to work on various free software projects[30]. The event starts with a kick-off in February.

In March mentoring organizations can starts submitting various task applications to GSoC. A mentoring organization is simply a running active free software project which releases software under any OSI approved license. The mentoring organizations publishes lists with various tasks for students to choose from. During the project the mentoring organization is supposed to monitor the progress of each student working on one of their tasks. At the end of the summer of code the monitoring organization is responsible for writing an evaluation about each participating student and then send these to Google.

In April students send in their applications for tasks to the mentoring organizations via the GSoC website. If the application is accepted the student will get to work on his or her task from April to September.

Google Summer of Code has helped progress in many free software project, after GSoC 2010 over 4500 GSoC tasks had been successful[30]. A few of these include:

- **FFMpeg** - Alexander Strange successfully added multithreading to the FFmpeg codec library as a GSoC project in 2008[26]. These changes were then merged into the main FFmpeg branch in March 2011[31].

- **Crystal Space** - Crystal Space had 6 different successful projects in GSoC 2010. The projects were deferred shading (Joseph Forte), automated progressive LOD generation and management (Eduardo Poyart), real time hair simulation and rendering (Alexandru-Teodor Voicu), new improved hardware augmented culler (Claudiu Mihail), advanced lighting for lighter2 (Mohit Taneja), integration of Recast and Detour into CEL and hierarchical pathfinding (both by Leonardo Domingues)[11].
• **Ubuntu** - Ubuntu also had a number of successful GSoC projects in 2010. Some of these includes Ubuntu One client for Android (Michal Karnicki), software centre improvements (Peter Gardenier) and Ubuntu One client for KDE SC users (Harald Sitter)[10].

6.3 Donations

Some projects are driven mostly by donations from the users of the applications. In these projects the donations either become the salary of the developers in the projects or are used to hire developers from outside of the project. Examples to projects driven partially by donations are Freenet and Ardour.

Freenet is an application used to create a “darknet” on top of the Internet which will let its users anonymously share and access files and websites published to Freenet. The Freenet project uses donations to hire one full-time developer to work on the project[24]. There are however also a large number of volunteers working on Freenet, but having a full time developer ensures that the project will keep on moving forward as long as there are donations. At the time of writing Freenet had 6500 USD in donations allocated to the paid developer, this means that Freenet has enough money to pay the developer for another 72 days of work[27]. Both companies and people are contributing to the Freenet project by donating, some notable donors includes Google and John Gilmore (one of the founders of the Electronic Frontier Foundation)[24].

Another project driven by donations is Ardour. Ardour is a feature-rich digital audio workstation released under the GPL in which one can record, edit and compose music and other audio. Ardour has, muck like Freenode, one paid developer. The developer paid to work on Ardour is its primary author Paul Davis[21] who receives most of the donations as his salary. There is no page on Ardour’s website clearly stating that Paul gets paid via the donations except for mentions about Paul working full-time as a developer of audio related free software and that donations makes full time development possible. I had before looking this up heard about how the primary author of Ardour did get paid via donations. I joined their IRC channel where three people, including the author Paul Davis himself confirmed this. Some of the money also goes to covering bug bounties (see the bullet issues below) and if the project gets more money than the goal for that month the excess will be sent to other contributors. Ardour has many different ways of contributing to the project by donating, these includes:

6.3.1 Subscribe

Donors can subscribe via PayPal to pay a monthly amount of money to Ardour. It is possible to select to donate 10, 4, 1 or 50 USD per month. Doing so helps the project greatly by giving it a stable source of money. It also helps by making it much easier to predict how much money that will be donated each month. The stability and predictability also helps the paid developer to plan for the future and decide whether or not he or she should keep working on the project as a full-time job even in the future.
6.3.2 Buying

At the download page for Ardour there is a text input field where one can choose any amount of money (even nothing) to donate when Ardour is downloaded. This is similar to how many proprietary software vendors sell their software except for the fact that users can decide on a price themselves. Alongside of pre-compiled packaged releases of the software one can also select to buy the source code (note that the GPL allows for selling software). The download page states that “If you are downloading a ready-to-run version, and choose to pay nothing, you will get one that is missing some handy functionality”[25]. This is another way to encourage users to donate at least something to the project. It also helps prevent users from donating only because they do not want to go through the donate procedures. One problem with his is that the software is also distributed by many other places (such as by package managers or other websites) so users of Ardour may never use the official download page even though they use the software.

6.3.3 Donate

One can choose to make a one time donation with any amount specified by donor. People who downloaded Ardour from other sources than the official web site can use this option instead of “buying” the software.

6.3.4 Issues

Another very interesting way of donating to Ardour is putting a bounty on issues in the issue tracker. When an issue has a bounty it means that the people solving the issue will receive the money attached to it as a reward[23]. For example, If company or person needs a specific feature or if a bug prevents the application from being useful to them they can attach a bounty to an issue describing the problem. That issue is then more likely to get attention and be resolved as the people solving it will get paid to do so. It does not automatically resolve all the issues, but it does help motivate developers to start working on the project and to get familiar with it.

At the time of writing the top three issues with the largest bounty are:

- MIDI editing/sequencer functionality in ardour - 26 bounties with a total reward of 1210 USD.
- AAF / OMF support - 10 bounties with a total reward of 455 USD.
- performance issue with lots of regions (bad scalability) - 1 bounty with a total reward of 100 USD.

6.4 University projects

Universities also have their share of contributions to free software projects. Some projects gets started in courses as assignments, projects or even degree projects such as Shaskel. Free software projects can also be developed as a part of research or because the university needs the software and decides to develop it themselves. Not all universities projects are released as free software, but as the software is often not created for profit it can make sense to release it as free software so that others may make use it.
There are quite a few project that started out, was or is developed as free software at universities, below are a few examples:

- **BSD** - Berkeley Software Distribution is a UNIX-based operating system developed by Berkeley university in California, USA. BSD was actively developed by Berkeley from 1979 through 1994. After the development ceased several other groups used the source code to continue building on the platform. There are even still today many projects based on BSD, including FreeBSD, NetBSD and OpenBSD. Apple’s Mac OS X is also partially based on BSD. [57]

- **Bluetile** - Bluetile is a tiling window manager designed to run in GNOME on Linux. It started as university project and thesis by Jan Vornberger at Carl von Ossietzky Universität in Oldenburg, Germany. [65]

- **OneSwarm** - OneSwarm is a privacy preserving peer-to-peer application designed to share data between its users. It was developed by people from the computer science and engineering department at the university of Washington in the USA. OneSwarm protect users’ privacy via the “friend-to-friend” method. “Friend-to-friend” means that the application only transfers data between the user and his or hers friends who in turn also only transfers data between their friends and so on. Because of that only the friends of a user knows what data he or she sent or requested. [13]

### 6.5 Fun, mastery and kindness

There are of course other reasons than for profit that people might choose to develop free software, below are a few examples.

- **Fun** - Software development is often compared to building with Lego. Software can, much like with Lego models, be created for fun. When software was created for fun it is sometimes released as free software so that it can be as useful as possible to others.

- **Mastery** - In the video “Drive” by Dan Pink and RSAnimate[62] mastery is described as a major reason as of to why free software is developed. Mastery is the drive of wanting to become better at something. By developing free software a person can gain more experience in software development thus getting better at it.

- **Kindness** - A person might see that there is a need for work to be done in some software project. Just like people might volunteer with charity work the same thing can be done in free software projects.

- **Need** - Developers might have a need for a certain application or feature and may therefore decide to implement it themselves. Afterwards it can be released as free software so that it can be as useful as possible to others.
7 Handling Issues

Handling issues is a large part of maintaining a free software project. “Issues” or “Tickets” are common terms for bug reports, feature requests, patches or other things that “needs to be done” in a software project. Issues are normally handled in issue trackers included as a part of a software management platform (see section 2).

7.1 Issue types

In a free software project anyone is usually able to new issues. These issues are then discussed with the project members and if they agree the issue might get resolved. How these issue are handled depends on what kind of issue it is.

7.1.1 Bug reports

A bug is an anomalous or erroneous behaviour in a system. When a user or tester experiences such a bug he or she may create a bug report and post that to the project’s issue tracker in the hopes that the problem will be resolved.

A bug report may first need to be confirmed. A bug report is confirmed when multiple people having the same bug simply posts a comment saying that they are also affected by the same bug. This does not always have to be the case if the bug is trivial. However, depending on what hardware, software or even software configuration the application runs on the user may or may not be affected by the bug. Because of these differences in the environment the bug may not be easy for others to reproduce. Describing how to reproduce a bug, how it affects the system and what the expected result was is essential for making it possible to solve it. If the bug is described in a sloppy or ambiguous way it may be classed as invalid and risks being closed before it is fixed.

When a bug is confirmed users and developers should attempt to find the cause of the bug. When this cause is found the developers will attempt to fix the problem. If a solution is found a patch is created and attached to the issue so that people affected by the bug can try it out themselves. If the patch works as expected it likely to be committed to the project repository.

7.1.2 Feature requests

Feature requests are requests for additional functionality in an application. When these are handled depends on how interesting or useful that particular feature appears to the project members.

The feature requests can also be used as a TODO list by the project members. An example to this is the Podcatcher [43] project where the project owner posts many of the future plans for the application as feature requests. A good thing about using the issue tracker as a TODO list is that the application users and other project members can get their say about what they think about a particular feature before it is implemented. Because of this feedback the feature requests might change, get closed or even implemented sooner as the project members can see just how interesting the feature is to the users. It is also possible to share implementation ideas or even attach patches which implements the said functionality.
7.1.3 Patches

A “patch” or “diff” is a file containing changes for an application. Patches are very similar to feature requests and bug reports except for that a solution already is attached to the issue.

Like with feature requests and bug reports, having patches publicly visible and having the possibility to discuss the patch can be of great help for a project. It could be that the patch causes regression in other features or solves the issue in an “ugly” way. The community can then help with improving the patch by discussion potential problems or by submitting their own versions. It also helps in the way that community members can try out the patch themselves before it is integrated into the project.

7.2 Getting issues resolved

Resolving issues can require a great deal of resources especially for larger projects such as Ubuntu. At the time of writing, Launchpad (section 2.1.5) tracks close to 700,000 issues across over 7150 projects and Ubuntu alone has over 87,000 open issues! These issues will not solve themselves but have to be solved manually by the various people involved in the projects.

It would be almost impossible to resolve all these issues without having enormous resources. As the projects hosted on Launchpad are free software, that enormous resource could be the community using and developing the software as everyone is allowed to help out with resolving issues. But how can one motivate the community to work on these issues? Ubuntu and other free software project have different ways of doing just that. This report will cover the Ubuntu Hug Day, the Ubuntu 5-A-Day and bounties.

7.2.1 Ubuntu Hug Day

The “Ubuntu Hug Day”, also called the “Ubuntu Bug Day”, is a recurring event in the Ubuntu community. The hug days are an attempt to focus the community on finding the cause (called “triaging”) for bugs in a specific area. These areas can either be applications such as Unity, Rhythmbox or Pidgin or other areas such as iso-image creation, package-less bugs or upgrades. Hug days usually occur about three to five times per month, each listing bugs within one specific area.

Bugs in hug days can either be new (meaning that no one has been working on the issue yet), incomplete (meaning that the issue lacks some essential information) or confirmed (meaning that the bug has been confirmed by multiple community members). The reason why the day is called a “hug day” is stated as follows on the Ubuntu Hug Day wiki page:

“The term Hug Day is a spin on Bug Day; every time someone triages a bug, then someone else should hug him/her. Why? This is a very special way for us to tell everyone that we love contributions! And triaging bugs is a really big contribution.”

7.2.2 Ubuntu 5-A-Day

Ubuntu 5-A-Day is a way to balance the workload while making sure that things actually gets done. The idea is that people involved in Ubuntu 5-A-Day should
each work on 5 bugs each day. To help motivate people further there are a lists with rankings giving credit to those who participate in Ubuntu 5-A-Day. The more issues that a person resolves, the better raking he or she will get.

It is made clear that everyone can help out and that one does not need to be a developer to do so. The wiki lists a number of things a person who would like to contribute can do. For non-developers:

- **Confirming** - Making sure that bugs listed in Launchpad actually are real bugs and then flag them as such.

- **Linking** - There are other issue trackers than Launchpad tracking issues for the same projects. An example to this is the issues for the Mozilla Firefox packages in Launchpad and issues for Firefox in Mozilla’s own issue tracker. Linking to upstream issue trackers means that if the two trackers both have the same issue, then there should be links from the issue in Launchpad to the issue in Mozilla’s tracker and vice versa. The links then makes it easy for users of one issue tracker to find the information located on the other one, this in turn can make it much easier to resolve both of the issues.

For developers:

- **Reviewing patches** - Making sure that patches attached to issues are correct and resolves the issue.

- **Triaging** - Finding the cause of a bug.

- **Assign packages** - Many Linux distributions uses packages and a package manager for installing, removing, upgrading and managing software in a system. For example, to install Mozilla Firefox one would ask the package manager to install the Firefox package instead of downloading it manually from Mozilla’s website. Each issue can specify that one or more packages are affected by that specific issue. This organizes the issue tracker by allowing people to browse all issues for a specific package and to see where they should look for the cause of the issue.

### 7.2.3 Bounties

As written about previously in section 6.3.4, bounties can be used to motivate people to resolve issues. “Bounties” means that people solving specific issues in various projects will receive an amount of money for doing so. There are a few different projects using bounties to motivate people to help out, these includes:

- **Ardour** - As written about previously in section 6.3.4, Ardour allows people to attach a bounty to issues in the issue tracker. The people who then resolve these issues will receive all the money attached to the issue.

- **Mozilla Firefox** - Mozilla has a program called “The Mozilla Security Bug Bounty Program”[41] which is designed to encourage security research in software developed by Mozilla. The program offers 3000 USD and a T-Shirt for those who solve a bug under certain criteria. The bug must be a remote exploit and previously unreported. While this will not resolve already existing issues, it does encourage creating and solving new ones.
- **Google Chrome** - Two days after the announcement of “The Mozilla Security Bug Bounty Program” Google announced that they would raise the reward for fixing critical security bugs in their browser Google Chrome[56]. Google used to offer 1337 USD for fixing critical bugs, after raising the bounty they instead offer up to 3133.70 USD.
8 Forks

A fork is a software project created by cloning the another project. The new project (the fork) is then developed independently of the original project. While source code and patches may still be exchanged between them, the projects can after a fork take completely different paths. It may even be so that one (or more) of the projects almost gets abandoned after a fork.

Distributed version control systems such as Git clones the entire repositories when they are checked out. Users are then encouraged to work on their local copies. When the local clone has been modified then either pull requests or patches are sent back to the project. While this is a forked repository it is not the kind of fork we will look at. We will look at completely forked projects, not just forked repositories.

We start by looking at a few reasons to why projects might get forked. After that we will take a look at a few different projects that have been forked, see why they were forked and what happened afterwards.

8.1 Reasons for forking

There are many different reasons for why a project would get forked, below are a few examples:

- **Company acquisition** - The company responsible for developing a project could have been acquired by another company. The new company can have different intentions for the project than what the old one had. The users and developers of the project may not agree to these new intentions and therefore decides to fork it. The new company will not have any control over the fork which will then not be affected by its intentions.

- **License change** - For various reasons a free software project may decide to switch to another license. The users and developers of the project may not agree to the new license and therefore decides to fork it while it still uses the old license. While the original project switches to a new license the fork can continue to be developed under the old one.

- **Merge disputes** - Developers of a project may not want to accept some changes made by other people. The reasons might be that the project members thinks that code is “ugly”, implemented in a bad way or they might just not like changes. The people who did not have their changes accepted could after that decide to fork the project. They will then be in control of the new project and will be free to apply whatever changes they desire.

- **Abandoned projects** - Sometimes free software projects are abandoned. It might be difficult to contact the original author or the author might not like the idea of someone else taking over the project. Developers who wish to continue working on the project will then have to fork it. Development can after that continue under another name.

- **Internal disputes** - Multiple developers can have different ideas of how a project should be run or how things should be implemented. This can lead to internal disputes which can in turn result in a fork. While both projects may end up with less resources all parties will be able to run their project as they please.
8.2 Forked projects

A lot of free software projects have at some point been forked. We are going to take a closer look at a few different projects to see why they were forked and what happened afterwards. The projects we will take a look at are XFree86 (forked into X.Org), Compiz (forked into Beryl) and OpenOffice.org (forked into LibreOffice).

8.2.1 XFree86 and X.Org

XFree86 and X.Org are both implementations of the X Window System which is a display manager for UNIX-based operating systems. XFree86 was forked into X.Org in April 2004 mostly due to an internal dispute[6]. The fork later on became popular because of a license change in XFree86.

One of the core developers of XFree86 (Keith Packard) was frustrated at its closed development model and decided to create the X.Org fork. Other XFree86 developers were not pleased with the project being forked. Shortly after that the creation of X.Org had been announced the XFree86 project announced that they would switch to a license (which was GPL-incompatible[64]).

XFree86’s decision to switch to a GPL-incompatible license was not well-received in the free software world. Many large Linux distributions and other UNIX-based operating systems announced shortly after the switch that they would abandon XFree86 and switch to the X.Org fork. The projects announcing the switch to X.Org included Red Hat, Debian, Gentoo and OpenBSD.

Today, XFree86 is mostly abandoned. XFree86 has not made a release since 2008[32] while X.Org is as of May 2011 alive and active[50]. X.Org is today used in almost every Linux distribution that makes use of the X Window System.

8.2.2 Compiz and Beryl

Compiz and Beryl are composited window managers for Linux. Being composited window managers means that they provide hardware accelerated graphical effects and animations to the desktop.

Compiz was forked into Beryl in September 2006[63] due to merge disputes. The author of Compiz did not like how changes submitted to the project were implemented and did not consider them to be of high quality. Beryl however accepted all these changes which made it more feature-rich than Compiz. Although forked, the projects remained relatively similar. Beryl gained traction fast probably because of its additional features.

Having two very similar projects allowed the developers to do as they pleased. But due to the fork the development became less focused and time was spent on transferring fixes and features between the two different projects.

On April the 4:th in 2007, less than a year after that Compiz had been forked, Jeffrey Laramie announced that the two projects would merge and once again form a single project[58]. The merge brought together the best parts from both Compiz and Beryl under the old name of Compiz.

8.2.3 OpenOffice.org and LibreOffice

OpenOffice.org and LibreOffice are free software office suites for Windows, Linux, MAC OS X and other UNIX-based operating systems. OpenOffice.org was forked
into LibreOffice by The Document Foundation in September 2010[59]. The Document Foundation was created in order to create a vendor-neutral governance body for the office suit shortly after that Oracle had acquired Sun. OpenOffice.org was then forked because Oracle was not transparent enough and refused to address various other management issues that bothered the community[61].

LibreOffice quickly gained traction as it was more community-driven than OpenOffice.org [60]. In an open letter published in November 2010 a group of over 30 OpenOffice.org contributors stated that they would join The Document Foundation. Oracle responded by forcing those who joined The Document Foundation to step down from their positions in the OpenOffice.org community council. By doing so Oracle had made it clear that contributors had to make a choice between OpenOffice.org and LibreOffice.

In April 2011 Oracle announced that they intended to discontinue the commercial development of OpenOffice.org. By that time many of the key developers and companies involved in developing OpenOffice.org (including Google, Canonical, Red Hat and Novell) had become members of The Document Foundation and were working on LibreOffice instead[61].
9 Profiting from free software

It might seem a bit strange to earn money on software that is free. But as described earlier in the introduction (section 1), the “free” in “free software” does not mean that it always is “free as in beer” (gratis). Instead, it means that the software is “free as in speech” (libre). Most of the free software is however also “free as in beer” (gratis), but that does not prevent one from earning money on it.

This section focuses on how to earn money on free software as a company. For information about profiting from free software as a project member, employee or in Google summer of code, see section 6.

To figure out how companies can earn money we will take a look at a free software-related company, Canonical which is the company behind Ubuntu. We will also look at how hardware manufactures can earn money using free software. But first we will look at how profiting from free software is affected by the various licenses.

9.1 Licenses

All licenses described in section 5 except for Creative Commons Non-Commercial (section 5.4) allow for commercial use of everything licensed under them. Selling free software is even encouraged by the Free Software Foundation which also states that selling free software is a good way to found development[14].

If the licenses had not allowed for using the software for commercial purposes, then it would not have been possible to sell support or hardware such as smartphones related to free software. In fact, non-commercial licenses are not even considered open source by the Open Source Initiative (OSI)[42], neither are they considered GPL compatible as they are more restrictive than the GPL.

Being able to use free software for commercial purposes is so important that projects such as Debian[37] and Fedora[39] do not allow software that forbid it to be distributed in their main repositories. The non-commercial software is then either not distributed at all or is distributed in special non-free repositories.

9.2 Relicensing

Software projects such as Qt and OGRE (version 1.6 or older) are earning money by selling relicensed versions of their software. When one buys the relicensed version of OGRE[38] they are free to make modifications to it without releasing these modifications to the public.

Trolltech used to use the relicensing method before it was acquired by Nokia. Trolltech allowed their Qt library to be used for free by free software projects while forcing proprietary ones to pay a licensing fee. They did this by using the GPL (section 5.2.1) for the free version of the library and a commercial license for the paid version.

This model works due to the nature of the GPL. As previously written in section 5.2.1, the GPL allows only for distributing GPL-licensed software together with other software that is licensed under an equally or more permissive license. Proprietary software vendors would then have to distribute the source code for their applications and allow its users to modify and redistribute it. If a proprietary software vendor does not agree to this but still wishes to uses the library it would have to use the paid version.
9.3 Services

Some companies have chosen to earn money on free software by selling services to both its users and other companies. In other words, they give away software while charging for services related to that same software. This section describes a few different paid services that various companies provide for a fee. I will use Canonical (the company behind Ubuntu) as an example to what kind of free software-related services companies can offer, both to other companies but also to regular consumers.

9.3.1 Support

Companies responsible for Linux distributions such as Red Hat and Canonical sell support as a major way of profiting from free software. The support is sold to both companies and consumers. The amount of money paid for the support depends on the service and for how long it is to be provided.

There are different levels of support and coverage for the different products that these companies are distributing. Below are a few examples on support sold by Canonical\[22\] for the Ubuntu Linux distribution:

- **Ubuntu Home Support** - Support provided to consumers for basic Ubuntu desktop usage. Covers installation, upgrades, applications and configuration. The support is online with a maximum response time of two business days. The price is about 90 UK Pounds for one year of support.

- **Ubuntu Advantage Standard Desktop** - Desktop support for business environments. Covers general business use of Ubuntu desktop such as e-mailing, browsing and messaging. Also includes the hosted management service called Landscape in which a company can manage multiple Ubuntu installations from a centralized web application. The price is 85 UK Pounds for 1 year and 170 UK Pounds for 3 years.

- **Ubuntu Advantage Advanced Server** - Total business support coverage for servers running Ubuntu. With this option companies can directly call Canonical at any time for complete direct support. As with the other Ubuntu Advantage services it also includes Ubuntu Landscape. The cost for Ubuntu Advantage Advanced Server is 760 UK Pounds for 1 year and 1930 UK Pounds for 3 years.

9.3.2 Storage

Canonical also attempts to earn money by offering users of Ubuntu online storage\[48\]. Ubuntu one is a service in which users can use to store their notes, bookmarks and personal files “in the cloud” and then sync them with multiple devices. The basic service is free but paying a monthly fee allows users to get addition storage and services such as streaming music stored in Ubuntu one to mobile devices.

The Ubuntu One service has a free software client and is integrated tightly with Ubuntu which makes it easy to start using. The Ubuntu One server software is however not free software and has not been released into the public at the time of writing\[49\].
9.3.3 Education

Another way to earn money on free software is offering education and training on how to configure and use it. Canonical offers several courses for both companies and regular consumers. A few of these courses are:

- **Ubuntu Desktop course** - This course can be taken as an online course or at one of the Ubuntu training partners. It teaches basic Ubuntu Desktop usage such as browsing the Internet, using office applications, playing games, desktop customization, viewing photos, watching videos, listening to music and so on. The course costs 33 UK Pounds.

- **Ubuntu Professional Training** - A course designed for those who deploy and administer Ubuntu in an office environment. Teaches installation, configuration, administration, using the command line and so on. The cost of the course is 220 UK Pounds.

- **Deploying Ubuntu Server** - A course for how to quickly and easily deploy Ubuntu Server across an organisation. Teaches advanced system administration and how to install and configure Ubuntu Server in an enterprise environment. The cost of this course is 10000 UK Pounds.

9.4 Hardware

There are many companies selling hardware which relies on large amounts of free software. There are many different types of hardware running free software such as cellphones, media centers, routers and so on. Below are a few devices which relies heavily on free software to function:

- **Nokia N900** - Nokia N900 is a smartphone (marketed as a “mobile computer”) running Maemo 5[40]. Maemo is a Debian-based GNU/Linux distribution aimed at smartphones and tablets. It is important to mention GNU here as other Linux-based operating systems (such as Android) may not make much use of the GNU tools. Maemo makes use of many free software technologies such as GTK, DBus, Gecko, Qt, X.org and Telepathy. Nokia N900 is not locked to Maemo but can also run other Linux-based distributions such as MeeGo.

- **TiVo Premiere** - TiVo Premiere is a media center which allows for the recording or streaming of both music and video. TiVo runs a modified version of Linux[15] and utilises many other free software projects including Busybox, OGG Vorbis, SQLite and BASH.

- **Dovado GR4** - The Dovado GR4 is a wired and wireless router with the ability to connect to the Internet via both a wired and a 3G connection (using a USB modem). If configured to, the router can use the 3G connection as a fall-back in case the wired connection goes down. The router runs Linux[8] and uses Busybox, curl and DNSMasq together with many other free software utilities.

The software shipped with the hardware might be under a license (such as the GPL in section 5.2) which requires the companies to provide the source code to said
software. This is often done by creating a web page where people can download archives containing the source code together with changes the company made to that source code. A link to that web page is often printed in the device’s manual together with the required license notices.

Although the source code and changes for the software is available, one might not actually be able to edit, compile and then run it on the hardware. This is by design and is called tivoization[5] (see section 5.2), the name actually comes from TiVo mentioned in the list above.
10 Shaskel

Shaskel is a free software game partially developed as a part of this degree project. Due to time constraints this report will only cover the initial setup and configuration of the project. The main goal with the Shaskel project is to give me additional experience in setting up and managing free software projects.

10.1 Setting up the project

To get Shaskel started a website and an IRC channel were created. Redmine was chosen for the website and is hosted at http://shaskel.furver.se/. Freenode was chosen to host the IRC channel #shaskel and Git (hosted at Furver using Gitolite) was chosen as the version control system.

10.1.1 Website

Redmine (http://www.redmine.org/) is a project management web application with issue tracker, blog, file hosting, wiki, forum, milestones and version control viewer. Redmine was chosen because it is very complete so no other separate web application was needed to manage the project. Some parts of Redmine such as the forum were a bit limited in functionality, it wasn’t for example possible to merge two forum topics into one. Redmine worked well without much problems even though it required some manual editing of its database to accomplish some tasks.

Installing and configuring Redmine was easy thanks to Ubuntu’s Debian packages. After installing the Redmine package all that was required was creating a new virtual host for it. http://redmine.furver.se/ was chosen to host Redmine while the index page of http://shaskel.furver.se/projects/shaskel/ redirects to http://redmine.furver.se/projects/shaskel/. http://shaskel.furver.se/ is also used for other data such as IRC logs.

For more information about Redmine see section 2.2.1 where it is described in-depth.

10.1.2 IRC Channel

Freenode was chosen to host the IRC channel which was registered using the name #shaskel. An IRC bot called “Rainbot” running the Supybot software was configured to log all activity in the channel. Having logs makes it easy for project members to read up on activities even if they weren’t present. The logs can be accessed from http://shaskel.furver.se/logs/ and are automatically updated when the bot writes them to the disc.

10.1.3 Version control system

Git was selected for the version control system. The repository was created using Gitolite which is a system for hosting multiple Git repositories with very flexible access control for multiple users on a single server account. Using Gitolite it is possible to for each user configure read or write permissions per repository or branch in the repositories. This means that users can get their own private branches to work on within the same repository.

Git daemon, Redmine and GitWeb were configured for read only access to the repository. Redmine and GitWeb both provide a web interface for the repository.
Using the web interfaces it is possible to inspect commits, files, branches and diffs in the Git repository. Using Redmine it’s also possible to reference individual commits in the wiki, forum or issues tracker. The web interfaces can also generate archives so that users can download the latest code without cloning or pulling from the repository. Git daemon allows for anonymous cloning of the repository using the Git protocol and the URL git://furver.se/shaskel/.

10.1.4 Backup

Because everything is hosted on my home server “Furver” a solid backup solution was required to not risk any data loss. A script was written which copies all the data related to the project into a single directory. The copied data includes the Redmine database (encrypted), files uploaded to redmine, IRC channel logs and the Git repository. The script is activated once every day at 05:00.

Every day at 06:00 TMM’s backup server “Milka” will connect to Furver using rsync over SSH. Milka will then download all changes from Furver and store them locally. The backups are also sometimes (once or twice each week) archived manually and uploaded to the Ubuntu One cloud service. The encrypted database ensures that no sensitive data about users will be leaked. The key for the database has been sent to several people to make sure that database can be decrypted and restored in case it becomes necessary.

10.2 Storyline

The storyline of Shaskel is based on the Inglip meme[51]. There were a few discussions about storyline details within the project before we settled. The game is named after Shaskel, a being of unknown powers which is the only one powerful enough to defeat the dark lord Inglip.

In the game you play as Lord Dentases and fight for lord Inglip using a modified version of the Agiores and the not-so-well known powers of Exicore. You will attempt to take back Castle Mashea and defeat Shaskel.

Shaskel is the last real threat to lord Inglip. With Shaskel out of the way and Castle Mashea reclaimed by its rightful owner (lord Inglip) we may finally be able to prevent end-time.

10.3 Finding contributors

In order to get people to join I started talking about the project with people I knew a few months before the project started. This was mainly to see whether or not people were interested in the project and if they were potential contributors.

It can be difficult to find out if people are being serious about contributing or if they are just generally like the idea. I found it useful to ask directly how serious they were about joining and contributing to the project. People generally liked the idea and a few people seemed interested in contributing.

5 people (TMM, AllegedlyDead, Club, Fisk and Myotonic) joined the project in the beginning and started contributing directly. They read my ideas, submitted their own, discussed decisions, learnt tools and showed others how to use them.
10.3.1 Reddit posts

AllegedlyDead suggested to post about the project to the Inglip subreddit[52]. As the Inglip subreddit is where Shaskel got its storyline from it was the perfect place to announce the project. We decided to wait until we had some documentation on the project website so that people could see that we were serious and already had done a bit of planning.

The post was posted to Reddit on the 17:th of February 2011[33]. The post invited people to join the project to help with the planning. The post was very well received and two people joined the IRC channel to discuss the project and possible contributions. But as expected, a week later the people who had joined had already left the IRC channel. As Inglip is a pretty small subreddit we did not expect a flood of people and there was no runnable code yet that could keep the interest up.

I decided to keep posting to Reddit about our progress, mainly to the Inglip subreddit but later also to larger subreddits. As the game becomes playable the interest in the game will most likely grow larger.

10.4 License

Shaskel uses GPLv3 (or later) as the license for game engine. Assets (game data) by default uses CC-BY-SA. We will accept all creative commons licensed assets not having the noncommercial constraint. It is also preferred (although not strictly required) that the license allows for modification of the data.

The reason why we prefer not to use assets with a noncommercial constraint is that they can not be distributed with some Linux distributions such as Fedora and Debian. There is still the possibility to ship these assets separately, but the game must be able to function without them.

The licenses were selected after a long discussion in the IRC channel on 2011-02-15.

10.5 Tools

Apart from using Git, Redmine and IRC we will need something to build the game with. At first we thought about using OGRE 3D for the graphics. However, after TMM took a closer look at OGRE 3D he discovered that it was depending on nVidia’s proprietary libCg library. Because OGRE 3D depended on a proprietary library we decided to scrap the idea of using it completely and not consider it an option.

TMM had heard about a project called Crystal Space. Crystal Space is a complete framework for developing 3D applications. While some features of Crystal Space indeed requires libCg those features are strictly optional. TMM discussed this issue in the Crystal Space channel and they seemed interested in dropping libCg completely in the future.

Blender was selected for 3D modelling as it is a mature free software applications. Crystal Space has good integration with Blender and provides plug-ins for exporting models from Blender to a Crystal Space native format.
11 Conclusion

While there is no one way of managing a free software project, there is a common pattern amongst the larger projects we have looked at. All of them have a project website, version control, and IRC channel. Additionally they have also had a forum of some sort (something we haven’t covered in detail), either as part of their project website or in the form of a mailing list. A similar setup should be sufficient for most free software projects. A license suitable for the goals of the project should also be selected, preferably one approved by the Free Software Foundation.

Judging by the amount of donations to Freenet and Ardour, it may be enough with only donations to make a living from a free software project for a single developer. However, for several developers it may be necessary to sell either licenses or hardware related to the project. Acquiring sponsors or being employed by a corporation to work on free software projects is also an option for individuals.
12 Future work

As only the initial set up of a free software project was done, starting up and managing a complete free software project would be one option for future work. This report does not go into any great detail about any of its topics and even skips some aspects such as mailing lists and project structures. Future work could include looking into any of those topics, or taking a closer look at topics already covered by this report. Future work could also include comparing different free software projects with each other and looking at how each are managed and how successful they are.
References


